Communicating Processes with Data for Supervisory Coordination

Jasen Markovski[*]
Department of Mechanical Engineering, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands,
j.markovski@tue.nl

We employ supervisory controllers to safely coordinate high-level discrete(-event) behavior of distributed components of complex systems. Supervisory controllers observe discrete-event system behavior, make a decision on allowed activities, and communicate the control signals to the involved parties. Models of the supervisory controllers can be automatically synthesized based on formal models of the system components and a formalization of the safe coordination (control) requirements. Based on the obtained models, code generation can be used to implement the supervisory controllers in software, on a PLC, or an embedded (micro)processor. In this article, we develop a process theory with data that supports a model-based systems engineering framework for supervisory coordination. We employ communication to distinguish between the different flows of information, i.e., observation and supervision, whereas we employ data to specify the coordination requirements more compactly, and to increase the expressivity of the framework. To illustrate the framework, we remodel an industrial case study involving coordination of maintenance procedures of a printing process of a high-tech Océ printer.

1 Introduction

Traditional software development techniques proved insufficiently flexible for development of quality control software, establishing the latter as an important bottleneck in design and production of complex high-tech systems [13]. This gave rise to supervisory control theory of discrete-event systems [20, 8] that studies automatic synthesis of (discrete-event) models of supervisory control software.

1.1 Supervisory Control

Supervisory controllers safely coordinate high-level system behavior, relying on observations made regarding the discrete(-event) system behavior by using sensory information, as depicted in Figure 1a). Based upon the observed signals, the supervisory controllers make a decision on which activities are allowed to be carried out safely, and send back control signals to the hardware actuators. By assuming that the supervisory controller reacts sufficiently fast on machine input, one can model this *supervisory control feedback loop* as a pair of synchronizing processes [20, 8]. The model of the machine, which is referred to as *plant*, is restricted by the model of the controller, referred to as *supervisor*. The synchronization of the supervisor and the plant, results in the *supervised plant*, which models the supervisory control loop, i.e., it specifies the behavior of the supervised system.

The activities of the machine are modeled as discrete events, whereas the supervisor is a process that synchronizes with the plant, and traditionally, it disables events by not synchronizing with them, whereas it enables events by synchronizing with them [20, 8]. As a result, the supervisor comprises the complete

[*] Supported by Dutch NWO project: ProThOS, no. 600.065.120.11N124.

Natallia Kokash and António Ravara (Eds.): 11th International Workshop on
Foundations of Coordination Languages and Self Adaptation (FOCLASA'12)
EPTCS 91, 2012, pp. 97–111, doi:10.4204/EPTCS.91.7

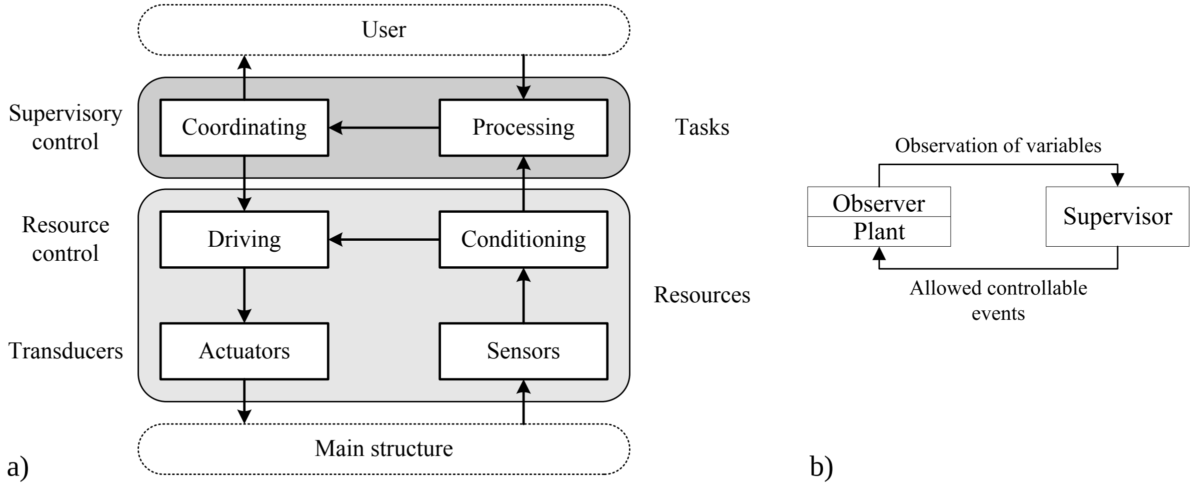

Figure 1: a) Supervisory control; b) Supervisory control feedback loop with data-based observations

history of the supervised system, i.e., it enumerates the state space of the supervised system [8, 3]. The events are split into *controllable events*, which can be disabled by the supervisor in order to prevent potentially dangerous or otherwise undesired behavior, and *uncontrollable events*, which must never be disabled by the supervisor. The former model activities over which control can be exhibited, like interaction with the actuators of the machine, whereas the latter model activities beyond the control of the supervisor, like observation of sensors or user interaction with the environment. Moreover, the supervised plant must also satisfy the *control requirements*, which model the safe or allowed behavior of the machine. In addition, it is typically required that the supervised plant is nonblocking, meaning that it comprises no deadlock and no livelock. To this end, every state is required to be able to reach a so-called *marked* or final state [20, 8], which denote states in which the plant is considered to have successfully completed its execution. The conditions that ensure the existence of such a supervisor are referred to as (nonblocking) *controllability* conditions [20, 8].

1.2 Motivation and Contributions

Our initial motivation for developing a process theory that distinguishes between the different flows of information between the plant and the supervisor is the oversimplification of the modeling of the supervisory control loop in the original proposal of [20, 8]. This manner of representation of this communication, by means of synchronizing action using automata-style synchronization, still prevails in modern state-of-the-art approaches, like [11, 9, 19, 24]. This is duely noted in [5], where a proposal is given to separate the different flows of information and to give a separate characterization of the process forms of the plant and the supervisor.

The approach investigated in [5] relies on propositional signals that stem from the states such that the supervisor has (intrinsic) knowledge regarding the state of the plant. Typically, state- or data-based approaches to supervisory control [14, 15, 8] require the use of an *observer*, which represents an addition to the plant as depicted in Figure 1b). The observer derives the state of the plant based on the history of observed events such that it can be directly communicated to the supervisor and employed for supervision. There are a couple of issues in the proposal of [5] when attempting to employ the process theory for modeling of supervisory control loops similar to the one depicted in Figure 1b). Namely, the semantics of the propositional signals relies on a predefined nondeterministic valuation effect function that updates the propositional signals based on the label of the taken transition and a set of possible future propositional signals [5]. This inevitably leads to unnecessary nondeterministically-chosen deadlock states when the intended propositional signal is not observed, making these deadlock states hard to interpret in a supervisory control setting. In addition, the observation of signals implicitly implies that the supervisor observes the state of the plant, not distinguishing between the plant and the observer. Admittedly, this is a standard practice, especially when modeling complex systems and development of compact and approachable models is of interest. Nonetheless, the underlying process theory should depict these nuances in a subtler manner.

To address the issues outlined above, we propose to replace and extend the propositional signals with variable assignments, which dynamically determine the valuation effect function resolving the first issue. As a welcome side effect, we obtain a compacter set of operational rules than the one presented in [5]. Moreover, by not having to implicitly couple the semantics of states with propositional signals, we have the option to model observers either as an intrinsic (integrated) part of the plant or as a separate process. In the setting of this paper, we rely on data-based observations, as depicted in Figure 1b). As discussed, the plant is augmented with an observer process, which may assign auxiliary data variables, based on the history of observed event. These data is required by the supervisor in order to make the correct control decision. We illustrate the situation by an example.

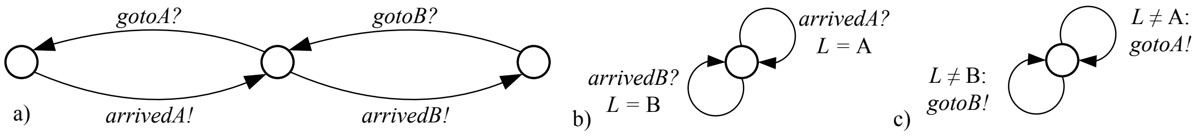

Figure 2: a) A plant that models the behavior of an automated guided vehicle; b) An observer that keeps track of the location of the vehicle of a); c) A supervisor that ensures proper coordination of the vehicle of a)

Example 1 *Let us assume that we have an automated guided vehicle that is capable of traveling to two positions, say A and B. We can issue two commands to the vehicle, namely* gotoA *and* gotoB*, so that the vehicle travels to A and B, respectively. When the vehicle arrives at the corresponding location, it reports back using the sensor signals* arrivedA *and* arrivedB*, respectively. We can model the behavior of this vehicle using the simple transition system, depicted in Figure 2a). Note that we distinguish between the direction of communicated commands and signals. By* event? *we denote a recipient party of the communication, and by* event! *we denote a sender party. We employ generic communication events, e.g.,* $\mathrm{event}!_2?_3$ *denotes a resulting communication event that occured between two sender and three recipient parties.*

Now, suppose that we wish to coordinate the movement of this vehicle, such that if the vehicle is at location X, then we do not issue a redundant command that sends the vehicle to the location X, for $X \in \{\mathrm{A}, \mathrm{B}\}$. Obviously, by employing only the transition system in Figure 2a) such coordination is not possible, since the model of the behavior of the vehicle does not comprise information regarding the location of the vehicle. To this end, we need an observer, which updates a variable, say L, with respect to the current location of the vehicle, as depicted in Figure 2b). The observer waits for confirmation signal, sent from the vehicles, that it has reached the corresponding location in order to update its status. By employing this location information, the supervisor can make the correct decision on which command is allowed to be issued.

Now, we can state the coordination or control requirements, based on the data observations as: if $L = \mathrm{X}$, then the event gotoX *should not be enabled, for $X \in \{\mathrm{A},\mathrm{B}\}$. By employing these control requirements, we can synthesize the supervisor depicted in Figure 2c). We note that the supervisor enables the movement commands, relying on guards that observe the shared location variable that is provided by the observer.*

To capture the controllability conditions involving the plant, the control requirements, and the supervisor we rely on a behavioral preorder termed *partial bisimulation*. In essence, we employ this preorder to state a relation between the supervised plant and the original plant allowing controllable events to be simulated, while requiring that uncontrollable event are bisimulated. This ensures that the supervisor does not disable uncontrollable events, while preserving the branching structure of the plant. Previous proposals like [11] and [19] rely on the process theory CSP [12], whereas other approaches rely on trace-based notions to capture controllability [9, 24]. In [11] the theory is extended with a specialized prioritized communication operator that captures the communication between the plant and the supervisor, later replaced by a refinement relation in failure semantics [19]. The control requirements depend on the observed data, and they are given in terms of global invariants that depend on the allowed data assignments, or they specify when a given event is allowed or disallowed, similarly to the informal specification of the control requirements in Example 1.

In the remainder of this paper, we first present the process theory and, thereafter, we discuss its application in a model-based systems engineering framework for supervisory coordination and control. To illustrate the framework, we revisit a case study that deals with coordination of maintenance procedures of a printing process of an Océ prototype printer [16]. The control problem is to synthesize a supervisory coordinator that ensures that quality of printing is not compromised by timely performing maintenance procedures, while interrupting ongoing print jobs as little as possible. Unlike previous attempts [5], we parameterize the model to handle multiple maintenance procedures concurrently. Due to confidentiality issues, we can only present an (obfuscated) part of the case study.

2 Communicating Processes with Data

To model data elements and guards, we extend the process theories $\mathrm{BSP}_{\|}$ of [3] and TCP* of [5], thus obtaining communicating processes with data. The result is a process theory encompassing *successful termination* that indicates final or marked states [20, 8], which model that the plant can successful terminate its execution; *generic communication action prefixes with data assignments*, which model activities of the plant and enable a dynamic valuation effect function; *guarded commands*, which condition transitions based on data assignments, and enable data observation and support supervision; *sequential composition*, which is an auxiliary operator required for definition of recursive processes; *iteration* to specify recurring behavior; and ACP-style *parallel composition with synchronization* [20] and *encapsulation*, which model together a flexible coupling in the feedback control loop based on given communication parties. We note that additional process operations can be easily added in the vein of [2, 5].

We remark that the synthesis tool Supremica [1], which we employ in the implementation of the model-based system engineering framework, supports the automata-like synchronization of [20, 8], which is standardly used in supervisory control theory. Moreover, there exists no distinction between sender and receiver parties in the parallel composition. The automata-like parallel composition synchronizes on all events from all processes that are in the common alphabet, whereas the remainder of the events is interleaved. It is not difficult to show, e.g., in the vein of [7], that our setting subsumes this parallel composition.

2.1 Syntax

In principle, we allow data elements to be of any type, given by the set D, even though only finite integer and enumerated types are currently supported by the synthesis tool [1]. By V, we denote the set of data variables, and by F, data expressions involving standard arithmetical operations supported by Supremica [1]. The arithmetical operations are evaluated with respect to $e\colon \mathsf{F} \to \mathsf{D}$. The guarded commands are given as Boolean formulas, whereas the atomic propositions are given by the predicates from the set $\{<,\leq,=,\neq,\geq,>\}$ and the logical operators are given by $\{\neg,\wedge,\vee,\Rightarrow\}$, denoting negation, conjunction, disjunction, and implication, respectively. We use B to denote the obtained Boolean expressions, which are evaluated with respect to a given valuation $v\colon \mathsf{B} \to \{\mathrm{false},\mathrm{true}\}$, where false denotes the logical value false, and true the logican value true. To this end, we update variables by a partial variable update function $f\colon \mathsf{V} \rightharpoonup \mathsf{D}$. The updating of variables is coupled with the action transitions that are labeled by actions from the set A. The set A is formed by all possible communication actions over a set of channels H, i.e, $\mathsf{A} \triangleq \{c!_m?_n \mid m,n \in \mathbb{N}, c \in \mathsf{H}\}$. We write $c!_n$ for $c!_n?_0$ and $c?_n$ for $c!_0?_n$ for $n \in \mathbb{N}$ and $c \in \mathsf{H}$, and we write $c!$ for $c!_1$ and $c?$ for $c?_1$. The set of process terms T is induced by T, given by:

$$T ::= 0 \mid 1 \mid a[f].T \mid \phi :\rightarrow T \mid \partial_H(T) \mid T+T \mid T \cdot T \mid T^* \mid T \parallel T$$

where $a \in \mathsf{A}$, $f\colon \mathsf{V} \rightharpoonup \mathsf{F}$, $\phi \in \mathsf{B}$, and $H \subseteq \mathsf{A}$. Each process $p \in \mathsf{T}$ is coupled with a global variable assignment environment that is used to evaluate the guards and keeps track of updated variables, notation $\langle p, (\alpha,\rho)\rangle \in \mathsf{T} \times \Sigma$ for $\Sigma = (\mathsf{V} \to \mathsf{F}) \times \mathsf{V}$. By $\alpha\colon \mathsf{V} \to \mathsf{F}$ we denote the assignment of the variables in order to consistently evaluate the guards, whereas the predicate $\rho \subseteq \mathsf{V}$ keeps track of the updated variables, which is needed for correct synchronization. We write $\sigma = (\alpha,\rho)$ for $\sigma \in \Sigma$, when the components of the environment are not explicitly required. The initial assignment $\sigma_0 = (\alpha_0, \mathrm{D}(\alpha_0))$, where $\mathrm{D}(f)$ denotes the domain of the function f, provides the initial values of all variables that the process comprises.

The theory has two constants: 0 denotes deadlock that cannot execute any action, whereas 1 denotes the option to successfully terminate. The action-prefixed process with variable update, corresponding to $a[f].p$, executes the action a, while updating the data values according to f, and continues behaving as p. The guarded command, notation $\phi :\rightarrow p$, specifies a guard $\phi \in \mathsf{B}$ that guards a process $p \in \mathsf{T}$. If the guard is successfully evaluated, the process continues behaving as $p \in \mathsf{T}$ or, else, it deadlocks. The encapsulation operator $\partial_H(p)$ encapsulates the process p in such a way that all communication actions in H that are considered as incomplete are blocked, so that the desired type of communication is enforced. For example, if we were to enforce communication between k processes over channel c, then $H = \{c!_m?_n \mid 0 < m+n,\ m+n \neq k\}$. The sequential composition $p \cdot q$ executes an action of the first process, or if the first process successfully terminates, it continues to behave as the second. The unary operator p^* represents iteration, or the Kleene star, that unfolds with respect to the sequential composition. The alternative composition $p+q$ makes a nondeterministic choice by executing an action of p or q, and continues to behave as the remainder of the chosen process. The binary operator $p \parallel q$ denotes parallel composition with generic communication actions, where the actions of both arguments can always be interleaved or, alternatively, communication can take place over common channels, keeping track of the number of involved sender and receiver parties.

2.2 Structural Operational Semantics

We give semantics in terms of labeled transition systems coupled with a environment that keeps track of the valuation of the data variables and the updated variables. The states of the labeled transition systems are labeled by the process terms themselves, and the dynamics of the process is given by a successful

$$\mathbf{1}\ \frac{}{\langle 1,\sigma\rangle\downarrow} \quad \mathbf{2}\ \frac{}{\langle a[f].p,(\alpha,\rho)\rangle \xrightarrow{a} \langle p,(\alpha\{\{X\mapsto e(f(X)) \mid X\in\mathrm{D}(f)\}\},\mathrm{D}(f))\rangle} \quad \mathbf{3}\,(\mathbf{4})\ \frac{\langle p,\sigma\rangle\downarrow}{\langle p+q,\sigma\rangle\downarrow}$$

$$\mathbf{5}\,(\mathbf{6})\ \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle}{\langle p+q,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle} \quad \mathbf{7}\ \frac{\langle p,\sigma\rangle\downarrow,\ \langle q,\sigma\rangle\downarrow}{\langle p\cdot q,\sigma\rangle\downarrow} \quad \mathbf{8}\ \frac{\langle p,\sigma\rangle\downarrow,\ \langle q,\sigma\rangle \xrightarrow{a} \langle q',\sigma'\rangle}{\langle p\cdot q,\sigma\rangle \xrightarrow{a} \langle q',\sigma'\rangle} \quad \mathbf{9}\ \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle}{\langle p\cdot q,\sigma\rangle \xrightarrow{a} \langle p'\cdot q,\sigma'\rangle}$$

$$\mathbf{10}\ \frac{}{\langle p^{*},\sigma\rangle\downarrow} \quad \mathbf{11}\ \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle}{\langle p^{*},\sigma\rangle \xrightarrow{a} \langle p'\cdot p^{*},\sigma'\rangle} \quad \mathbf{12}\ \frac{\langle p,\sigma\rangle\downarrow,\ \langle q,\sigma\rangle\downarrow}{\langle p \parallel q,\sigma\rangle\downarrow} \quad \mathbf{13}\,(\mathbf{14})\ \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle}{\langle p \parallel q,\sigma\rangle \xrightarrow{a} \langle p' \parallel q,\sigma'\rangle}$$

$$\mathbf{15}\ \frac{\langle p,\sigma\rangle \xrightarrow{c!_{k}?_{\ell}} \langle p',(\alpha',\rho')\rangle,\ \langle q,\sigma\rangle \xrightarrow{c!_{m}?_{n}} \langle q',(\alpha'',\rho'')\rangle,\ \alpha'|_{\rho'\cap\rho''} = \alpha''|_{\rho'\cap\rho''}}{\langle p \parallel q,\sigma\rangle \xrightarrow{c!_{k+m}?_{\ell+n}} \langle p' \parallel q',(\alpha'\{\alpha''|_{\rho''\setminus\rho'}\},\rho'\cup\rho'')\rangle} \qquad \mathbf{16}\ \frac{\langle p,\sigma\rangle\downarrow,\ v(\phi)=\text{true}}{\langle \phi :\rightarrow p,\sigma\rangle\downarrow}$$

$$\mathbf{17}\ \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle,\ v(\phi)=\text{true}}{\langle \phi :\rightarrow p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle} \quad \mathbf{18}\ \frac{\langle p,\sigma\rangle\downarrow}{\langle \partial_H(p),\sigma\rangle\downarrow} \qquad \mathbf{19}\ \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle,\ a\notin H}{\langle \partial_H(p),\sigma\rangle \xrightarrow{a} \langle \partial_H(p'),\sigma'\rangle}$$

Figure 3: Operational rules

termination option predicate $\downarrow\ \subseteq \mathsf{T}\times\Sigma$, that plays the role of final or marked states for nonblocking supervision [20, 8], and an action transition relation $\longrightarrow\ \subseteq (\mathsf{T}\times\Sigma)\times\mathsf{A}\times(\mathsf{T}\times\Sigma)$. We write $\langle p,\sigma\rangle\downarrow$ for $\langle p,\sigma\rangle\in\ \downarrow$ and $\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle$ for $(\langle p,\sigma\rangle,a,\langle p',\sigma'\rangle)\in\ \longrightarrow$.

To present concisely the update of the assignments, we introduce several auxiliary operations. We write $f|_C$ for the restriction of the function f to the domain $C\subseteq\mathrm{D}(f)$, i.e., $f|_C \triangleq \{x\mapsto f(x) \mid x\in C\}$. Also, we introduce the notation $f\{f_1\}\ldots\{f_n\}$, where $f\colon A\to B$ and $f_i\colon A\rightharpoonup B$ for $1\le i\le n$ are partial functions with mutually disjoint domains, i.e., $\mathrm{D}(f_i)\cap\mathrm{D}(f_j)=\emptyset$ for $i\neq j$. For every $x\in A$, we have that $f\{f_1\}\ldots\{f_n\}(x) = f_j(x)$, if there exists some j such that $1\le j\le n$ and $x\in\mathrm{D}(f_j)$, or $f\{f_1\}\ldots\{f_n\}(x) = f(x)$ otherwise. We define $\downarrow$ and $\longrightarrow$ using structural operational semantics [2], depicted by the operational rules in Figure 3. We note that symmetrical rules are not depicted, and their number is only denoted in brackets next to the number of the rule that is to be applied for the process on the left side of the operation.

Rule 1 states that the termination constant has the option to successfully terminate. Rule 2 states that the action prefix enables action transitions, whereas the target assignment must be updated accordingly. Namely, the variables in the domain of the partial variable assignment function are updated with the evaluated data expression. Rules 3 and 4 state that the alternative composition can successfully terminate if one of its summands has the option to successfully terminate. Similarly, action transitions are possible in the alternative composition if one of its summands can perform them, as given by rules 5 and 6. Rule 7 states that the sequential composition has a termination option if both of its components have a termination option. If the first component terminates, then the sequential composition continues behaving as the second component, as given by rule 8. If the first component can perform an action transition, then the target process sequentially composes the target of the action transition of the first component with the second component, as given by rule 9. Iteration always has a termination option as given by rule 10, because of properties of composition of recursive processes [2]. Rule 11 shows that iteration unfolds with respect to sequential composition. Rule 12 states that the parallel composition can successfully terminate only if both of its components have successful termination options. Rules 13 and 14 enable interleaving in the parallel composition, even interleaving of transitions that stem from the same channel. Synchronizing of action transitions is possible for action that stem from the same channel as depicted by rule 15. The resulting communication action must account for the accumulative number of sender and receiver communication parties. The sets ρ' and ρ'' identify the updated variables, so the common updated variables are given by $\rho' \cap \rho''$. The target environment updates the target environment of the first component with the remaining target environment of the second component, which can also be done symmetrically with respect to the second component. Rules 16 and 17 state that if the propositional guard is successfully evaluated, then the guarded command can successfully terminate or perform an action transition, respectively, provided that the guarded process can do so. Rule 18 states that successful termination is not affected by the encapsulation operator, whereas rule 19 states that all actions in the parameter set $H \subseteq \mathsf{A}$ are blocked.

2.3 Partial Bisimulation

The behavioral relation that we employ is an extension of partial bisimulation [3], which is able to handle data and variable assignments. Here, we directly employ the approach of [6, 2], where this extension is shown for bisimulation.

A relation $R \subseteq \mathsf{T} \times \mathsf{T}$ is said to be a partial bisimulation with respect to a bisimulation action set $B \subseteq \mathsf{A}$, if for all $(p,q) \in R$ and $\sigma \in \Sigma$, it holds that:

1. $\langle p, \sigma \rangle \downarrow$ if and only if $\langle q, \sigma \rangle \downarrow$;
2. if $\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle$ for $a \in \mathsf{A}$, then there exist $q' \in \mathsf{T}$ and $\sigma' \in \Sigma$ such that $\langle q, \sigma \rangle \xrightarrow{a} \langle q', \sigma' \rangle$ and $(p', q') \in R$;
3. if $\langle q, \sigma \rangle \xrightarrow{b} \langle q', \sigma' \rangle$ for $b \in B$, then there exist $p' \in \mathsf{T}$ and $\sigma' \in \Sigma$ such that $\langle p, \sigma \rangle \xrightarrow{b} \langle p', \sigma' \rangle$ and $(p', q') \in R$.

If R is a partial bisimulation relation such that $(p,q) \in R$, then p is partially bisimilar to q with respect to B and we write $p \preceq_B q$. If $q \preceq_B p$ holds as well, we write $p \leftrightarrow_B q$.

It is not difficult to show that partial bisimilarity is a preorder for the process terms in T [7] following the guidelines of [6]. In addition, following the guidelines of [21], it can be shown that $\preceq_B$ is a partial bisimulation relation with respect to $B \subseteq \mathsf{A}$. Thus, we obtain the partial bisimulation preorder and equivalence, similarly as for simulation preorder and equivalence [10]. Moreover, the partial bisimulation preorder can be shown a precongruence for the considered processes operations following the guidelines of [7, 3], where a suitable extension to the tyft format for structural operational semantics with data of [18] is proposed. Consequently, the partial bisimulation equivalence is a congruence, which enables us to build a standard term model using the quotient algebra modulo $\leftrightarrow_B$ in the vein of [2]. Finally, we note that $p \leftrightarrow_{\mathsf{A}} q$ amounts to bisimulation [3], whereas $p \preceq_{\emptyset} q$ reduces to simulation preorder [3] and $p \leftrightarrow_{\emptyset} q$ reduces to simulation equivalence [3] for processes with data [2, 10].

3 A Process-Theoretic Approach to Supervisory Coordination

First, we characterize the process terms that can be used to specify the plant and the supervisor. Thus, we distinguish between the two different flows of information on syntactic level. We employ the notion of partial bisimulation to define the relationship between the plant and the supervisor in order to ensure that the supervisor does not disable any uncontrollable events. Thereafter, we identify a set of data-based control requirements that are typically employed in specification documents. Finally, we describe the model-based system engineering framework and we discuss its implementation.

3.1 Plant and Supervisor Syntax

We distinguish between controllable and uncontrollable actions transitions that stem from the sets of controllable H_C and uncontrollable H_U channels, where $\mathsf{H}_\mathsf{C} \cap \mathsf{H}_\mathsf{U} = \emptyset$ and $\mathsf{H}_\mathsf{C} \cup \mathsf{H}_\mathsf{U} = \mathsf{H}$. We put $\mathsf{C} \triangleq \{c!_m?_n \mid m,n \in \mathbb{N},\ c \in \mathsf{H}_\mathsf{C}\}$ and $\mathsf{U} \triangleq \{u!_m?_n \mid m,n \in \mathbb{N},\ u \in \mathsf{H}_\mathsf{U}\}$. We model marked states by adding a successfully termination option to the corresponding state. We note that in the process theoretic setting, successful termination plays an additional role of enabling the sequential composition of processes [2, 5], which is not present in the automata theory of [20, 8]. We restrict the syntax of the plant and the supervisor, given by P and S, respectively, as follows:

$$\begin{aligned} P &::= 0 \mid 1 \mid c?_n[f].P \mid u!_m?_n[f].P \mid \phi :\to P \mid \partial_H(P) \mid P+P \mid P \cdot P \mid P^* \mid P \parallel P \\ S &::= 1 \mid c![\emptyset].S \mid S+S \mid \phi :\to S \mid S^*, \end{aligned} \tag{1}$$

where $c \in \mathsf{H}_\mathsf{C}$, $u \in \mathsf{H}_\mathsf{U}$, $f\colon \mathsf{V} \rightharpoonup \mathsf{F}$, $m,n \in \mathbb{N}$, $\phi \in \mathsf{B}$, and $H \subseteq \mathsf{A}$. We note that we specify a monolithic supervisor, i.e., the supervision is executed by a single process. For modular or distributed supervision [8], the syntactic form of the supervisor from (1) should be adjusted appropriately, so that it can admit several concurrent communicating processes.

We require the supervisor to be a deterministic process [3], which sends feedback to the plant in terms of synchronizing controllable events, and it does not alter the state of the plant in any other way, i.e., it comprises no variable assignments. The supervisor relies on data observation from the plant to make supervision decisions in the vein of [17]. Thus, the supervisor observes the state of the plant, identified by the values of the (shared) variables, and enables controllable events by synchronizing with a corresponding sender event. It does not influence uncontrollable events, so they are safely interleaved in the communication with the plant. Consequently, the supervisor does not have to keep a history of events, so it can be also be defined as an iterative process, which observes assigned data by employing guarded commands. This alternative definition is given as:

$$s = \left(\textstyle\sum_{c \in \mathsf{H}_\mathsf{C}} \phi_c :\to c![\emptyset].1 + \psi :\to 1\right)^*, \tag{2}$$

where $\phi_c, \psi \in \mathsf{B}$ for $c \in \mathsf{H}_\mathsf{C}$. A supervisor of form (2) employs data value observation to identify the state of the plant and send back feedback regarding controllable events by synchronizing on self loops, as specified by $\sum_{c \in \mathsf{H}_\mathsf{C}} \phi_c :\to c![\emptyset].1$. It can potentially disable undesired termination options in states identified by $\psi \in \mathsf{B}$. The guards ϕ_c for $c \in \mathsf{H}_\mathsf{C}$ and ψ depict the supervision actions [17].

3.2 Supervised Plants and Controllability

If we suppose that the plant is given by $p \in P$ and the supervisor is given by $s \in S$, then the supervised plant can be specified as $\partial_H(p \parallel s)$ in general, where the encapsulation enforces desired communication and the set $H \subset \mathsf{A}$ comprises unfinished communication events, which differ per case. To ensure that no uncontrollable events are disabled by the supervisor, we employ partial bisimilarity to provide a relation between the supervised and the original plant. We note, however, that most of the other approaches, like [20, 8, 11, 24] to name a few, employ synchronizing actions, where two transitions with the same label synchronize in a resulting transitions, which is again labeled by that same label. In that case, the relation can be provided directly as in [3], because the labels of the transitions in the supervised and the original plant coincide.

In the setting of this paper, however, we have to rename certain actions in the original plant so that we can mimic the presence of a supervisor, which is necessitated in order to make the plant operational.

$$\mathbf{20}\ \frac{\langle p,\sigma\rangle\!\downarrow}{\langle\xi(p),\sigma\rangle\!\downarrow}\qquad \mathbf{21}\ \frac{\langle p,\sigma\rangle \xrightarrow{c?_n[f]} \langle p',\sigma'\rangle,\ c\in \mathsf{H_C}}{\langle\xi(p),\sigma\rangle \xrightarrow{c!?_n[f]} \langle\xi(p'),\sigma'\rangle}\qquad \mathbf{22}\ \frac{\langle p,\sigma\rangle \xrightarrow{u!_m?_n[f]} \langle p',\sigma'\rangle,\ u\in \mathsf{H_U}}{\langle\xi(p),\sigma\rangle \xrightarrow{u!_m?_n[f]} \langle\xi(p'),\sigma'\rangle}$$

Figure 4: Renaming operation that renders the controllable communication events of the plant completed

$$\mathbf{23}\ \frac{\langle p,\sigma\rangle \models \neg\phi \Rightarrow \not\xrightarrow{a}}{\langle p,\sigma\rangle \models \xrightarrow{a} \Rightarrow \phi}\qquad \mathbf{24}\ \frac{v(\phi)=\text{false}}{\langle p,\sigma\rangle \models \phi \Rightarrow \not\xrightarrow{a}}\qquad \mathbf{25}\ \frac{\langle p,\sigma\rangle \not\xrightarrow{a}}{\langle p,\sigma\rangle \models \phi \Rightarrow \not\xrightarrow{a}}\qquad \mathbf{26}\ \frac{v(\phi)=\text{true}}{\langle p,\sigma\rangle \models \phi}$$

Figure 5: Satisfiability of data-based control requirements

To this end, we employ a specific partial renaming operation $\xi : \mathsf{T} \mapsto \mathsf{T}$ that renders the controllable communication actions of the plant as completed. This is in accordance with the syntax of the plant and the supervisor specified in (1), since the plant must wait for an enabling control signal for every controllable event. The operational rules that define the renaming operation ξ are given in Figure 4.

Now, we can specify the relation between the supervised and the original plant as:

$$\partial_H(p \parallel s) \preceq_{\mathsf{U}} \xi(p). \tag{3}$$

It states that the supervised plant has controllable events enabled by the supervisor that can be simulated by the original plant in which all controllable events have been enabled, whereas no uncontrollable events can be disabled. We note that, in the setting of this paper, one can observe that the syntactical restrictions imposed on the supervisor actually imply this relation.

It is not difficult to show, again in the vein of [21, 3, 7], that the traditional notions of language-based controllability of [20, 8] for deterministic system and state controllability [17, 9, 24] for nondeterministic systems are implied by (3).

3.3 Data-Based Control and Coordination Requirements

In the setting of this paper, we consider data-based control and coordination requirements, which are stated in terms of boolean expressions ranging over the data variables, and may additionally specify which events are allowed with respect to the observed data values. For a setting with event-based control requirements, we refer the interested reader to [3], whereas for state-based control requirements, a preliminary investigation is given in [5]. The data-based control requirements, denoted by the set R, have the following syntax induced by R:

$$R ::= \ \xrightarrow{a} \Rightarrow \phi \ \mid \ \phi \Rightarrow \not\xrightarrow{a} \ \mid \ \phi,$$

for $a \in \mathsf{A}$ and $\phi \in \mathsf{B}$. A given control requirement $r \in \mathsf{R}$ is satisfied with respect to the root of the process term $p \in \mathsf{T}$ in the assignment environment $\sigma \in \Sigma$, notation $\langle p,\sigma\rangle \models r$, according to the operational rules depicted in Figure 5. By $\langle p,\sigma\rangle \not\xrightarrow{a}$ we denote that $\{\langle p',\sigma'\rangle \mid \langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle\} = \emptyset$.

The first form of control requirements is introduced for modeling convenience as a frequently occurring case [15] and it is equivalent to the second form, as given by rule 23. Rule 24 states that if the state does satisfy the data assignment, then the requirement is trivially satisfied. Rule 25 states a

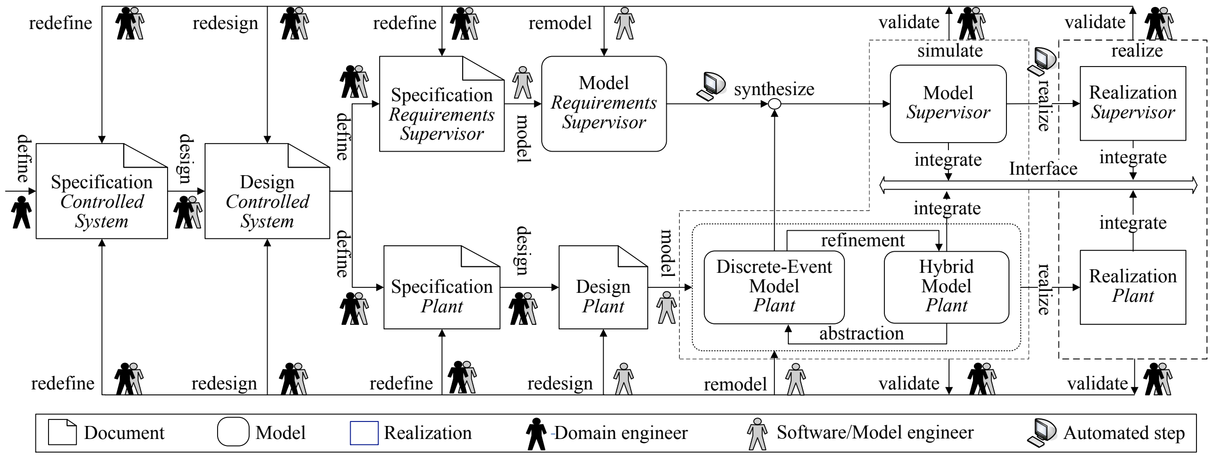

Figure 6: Model-based systems engineering framework for supervisory controller synthesis

so-called state-transition exclusion requirement [15], which is satisfied if no transition with the excluded label is possible. Rule 26 states that a state-exclusion requirement restricts the states with the given data assignments, thus disabling unsafe or forbidden states, and must be upheld in every state. To ensure that the control requirements are globally satisfied, we extend $\models$ to $\models^*$, which requires that the control requirements are satisfied for every reachable state. To this end, we first define a trace transition relation $\langle p,\sigma\rangle \xrightarrow{t}{}^{*} \langle p',\sigma'\rangle$ for some $t = a_1 \ldots a_n \in \mathsf{A}^*$. If $n = 0$ we have the empty trace $t = \varepsilon$ with $\langle p,\sigma\rangle \xrightarrow{\varepsilon}{}^{*} \langle p,\sigma\rangle$, whereas if $n > 0$, then we have $\langle p,\sigma\rangle \xrightarrow{a_1} \langle p_1,\sigma_1\rangle \xrightarrow{a_2} \langle p_2,\sigma_2\rangle \xrightarrow{a_3} \ldots \xrightarrow{a_n} \langle p',\sigma'\rangle$ for some $p_1,\ldots,p_{n-1} \in \mathsf{T}$, $\sigma_1,\ldots,\sigma_{n-1} \in \Sigma$, and $a_1,\ldots,a_{n-1} \in \mathsf{A}$. Now, we define that $p \models^* r$ if $q \models r$ for every $p' \in \mathsf{T}$ such that $\langle p,\sigma\rangle \xrightarrow{t}{}^{*} \langle p',\sigma'\rangle$ for $\sigma,\sigma' \in \Sigma$ and $t \in \mathsf{A}^*$.

To ensure that the supervised plant respects the data-based control requirements, given by $R \subset \mathsf{R}$, we require that for the initial variable assignment $\sigma_0 \in \Sigma$ it holds that

$$\langle p \parallel s, \sigma_0\rangle \models^* \textstyle\bigwedge_{r\in C} r. \tag{4}$$

In addition, a nonblocking supervisor must ensure that every state in the supervised plant can reach a state that has a successful termination option, i.e., for every $\langle p',\sigma'\rangle \in \mathsf{T} \times \Sigma$ and $t \in \mathsf{A}^*$ such that $\langle p \parallel s,\sigma_0\rangle \xrightarrow{t}{}^{*} \langle p',\sigma'\rangle$, there exists $\langle p'',\sigma''\rangle \in \mathsf{T} \times \Sigma$ and $t' \in \mathsf{A}^*$ such that $\langle p',\sigma'\rangle \xrightarrow{t'}{}^{*} \langle p'',\sigma''\rangle$ and $\langle p'',\sigma''\rangle\downarrow$ holds.

3.4 Model-Based Systems Engineering Framework

To structure the process of supervisory control synthesis we employ the framework depicted in Figure 6 [22, 15, 4]. The modeling process begins with an informal specification of the controlled system, i.e., the desired product, written by domain engineers. A design of the controlled system follows, contrived by domain and software engineers together. The design most importantly defines the modeling level of abstraction and the control architecture. Subsequently, it is used to separate the plant and the control requirements, a joint task of domain and software engineers. Here, a decision is made to which extent the control is managed by the software, and which part is implemented in hardware. The resulting informal documents specify the plant and control requirements, respectively. In the following, we omit the roles of the engineers as they are clear from the context.

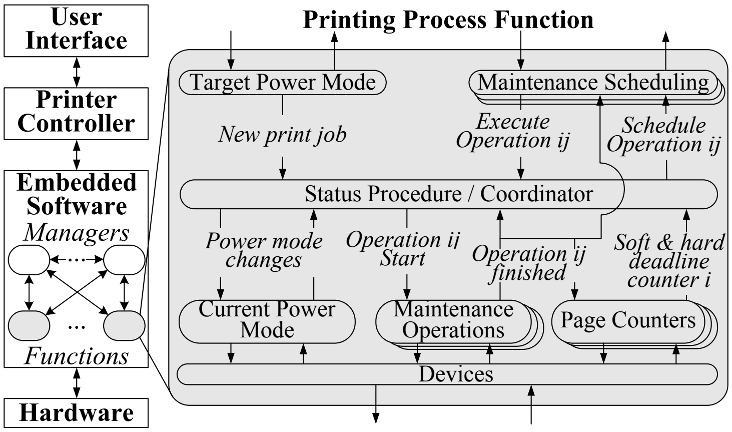

Figure 7: Modeling of the printing process function

Most plants typically contain (continuous) hybrid behavior, whereas supervisor synthesis requires a discrete-event abstraction. The hybrid model is suitable for simulation purposes, and it can be abstracted to a discrete-event model for synthesis purposes [20, 8]. Alternatively, a discrete-event model can be made, and subsequently refined [23]. In the design of the plant, decisions are made on the level of abstraction that is used, and what is significant discrete-event and hybrid behavior. In parallel, a model of the control requirements is made following the specification documents. The discrete-event model of the plant, together with the model of the control requirements, are input to the synthesis tool, which automatically synthesizes a supervisor.

Software-in-the-loop simulation is used to validate the supervisor coupled with a hybrid model of the plant, and hardware-in-the-loop simulation can be used to validate the supervisor against a prototype of the plant. If the validation is not satisfactory, the control requirements and/or the plant model need to be remodeled or redefined. In certain cases, a complete revision proves to be necessary, which might even require redefining the specification of the whole controlled system. Finally, the control software is generated automatically, based on the validated models. Note that software engineers in the framework act more as ‘model’ engineers, shifting their focus from writing code to modeling.

We opt for Supremica [1] as a synthesis tool because it provides the greatest modeling convenience and range of options with respect to specifying plants with data and optimized synthesis procedures [17]. We remark that the state-of-the-art synthesis tools support the prevailing automata-style specifications and composition [1, 8]. To be able to execute industrial case studies, we are impelled to translate the original process-algebraic specification to an input accepted by the tool.

4 Coordinating Maintenance Procedures of a Printing Process Function

An abstract view of the control architecture of a high-tech printer is depicted in Figure 7. Print jobs are sent to the printer by means of the user interface. The printer controller communicates with the user and assigns print jobs to the embedded software, which actuates the hardware to realize print jobs. The embedded software is organized in a distributed way, per functional aspect, such as, paper path, printing process, etc. Several managers communicate with the printer controller and each other to assign tasks to functions, which take care of the functional aspects.

We depict a printing process function comprising several maintenance operations in Figure 7. Each

$$\begin{aligned}
&CPM \triangleq \big(Stb2Run?\{CPM \mapsto 2\}.\_InRun\{CPM \mapsto 3\}.Run2Stb?\{CPM \mapsto 4\}.\_InStb\{CPM \mapsto 1\}.1+1\big)^{*}\\
&MO_{ij} \triangleq \big(OpStart_{ij}\{MO_{ij} \mapsto 2\}.\_OpFin_i!\{MO_{ij} \mapsto 1\}.1+1\big)^{*}\\
&PC_i \triangleq \big(\_SoftDln_i\{PC_i \mapsto 2\}.\big(\_HardDln_i\{PC_i \mapsto 3\}.\_OpFin_i?\{PC_i \mapsto 1\}.1+\_OpFin_i?\{PC_i \mapsto 1\}.1\big)+\\
&\qquad\quad \_OpFin_i?.1+1\big)^{*}\\
&MS_i \triangleq \big(SchOper_i?\{MS_i \mapsto 2\}.\_ExOper_i\{MS_i \mapsto 3\}.\_OpFin_i?\{MS_i \mapsto 1\}.1+1\big)^{*}\\
&TPM \triangleq \big(\_NewJob\{TPM \mapsto 2\}.\_JobFin\{TPM \mapsto 1\}.1+1\big)^{*}\\
&PPF \triangleq \partial_{\{\_OpFin_i?,\_OpFin_i?_2,\_OpFin_i!?\}}\Big(CPM \parallel \big(\|_{i\in I} MS_i\big) \parallel \big(\|_{j\in J_i,\, i\in I} MO_{ij}\big) \parallel \big(\|_{i\in I} PC_i\big) \parallel TPM\Big)
\end{aligned}$$

Figure 8: Process-algebraic specification of the plant

function is hierarchically organized to: (1) controllers: Target Power Mode and Maintenance Scheduling, which receive control and scheduling tasks from the managers; (2) procedures: Status Procedure, Current Power Mode, Maintenance Operation, and Page Counter, which handle specific tasks and actuate devices, and (3) devices as hardware interface. Status Procedure is responsible for coordinating the other procedures given the input form the controllers. The control problem is to synthesize a supervisory coordinator that ensures that quality of printing is not compromised by timely performing maintenance procedures, while interrupting ongoing print jobs as little as possible. We specify the coordination rules that ensure safe behavior of the system below.

4.1 Process-Algebraic Specification

We briefly describe the procedures that comprise the plant, whose process-algebraic specification is given in Figure 8. We assume that the page counters are indexed by the set I, whereas for each counter i there are J_i maintenance procedures to be triggered. Also, labels of uncontrollable events begin with an underscore. Furthermore, we identify states by means of variable observation, i.e., we incorporate the observer inside the plant specification, so we assign the variables MO_{ij} to Maintenance Operation ij, PO_i to Page Counter i, MS_i to Maintenance Scheduling i, TPM to Target Power Mode, and CPM to Current Power Mode for $j \in J_i$ and $i \in I$. Initially, the variables are set to 1, which identifies the first state. The plant model is depicted in Figure 8, where Printing Process Function is defined by PPF and $\|_{p\in P}\, p$ denotes the parallel composition of the processes in P.

Current Power Mode sets the power mode to run or standby depending on the enabling signals (*Stb2Run* and *Run2Stb*) from Status Procedure, and sends back feedback by employing *_InRun* and *_InStb*, respectively. Maintenance Operation ij for $j \in J_i$ and $i \in I$ either carries out a maintenance operation, started by $\_OpStart_{ij}$ or it is idle. The confirmation is sent back by the events $\_OpFin_{ij}!$ for $j \in J_i$ and $i \in I$, which synchronize with Maintenance Scheduling and Page Counter. Page Counter i for $i \in I$ counts the printed pages since the last maintenance and sends signals $\_SoftDln_i$ and $\_HardDln_i$, when soft or hard deadlines are reached, respectively. It is responsible for the set of maintenance procedures in J_i. A soft deadline signals that maintenance should be performed, but it is not yet compulsory if there are pending print jobs. A hard deadline is reached when maintenance of the printing process must be performed to ensure quality of the print. The page counter is reset, triggered by the synchronization on $\_OpFin_{ij}?$, each time that maintenance is finished. The controller Target Power Mode sends signals regarding incoming print jobs to Status Procedure by *_NewJob*, which should set the printing process to run mode for printing and standby mode for maintenance and power saving. When the print job is finished, the signal *_NoJob* is sent. Maintenance Scheduling i for $i \in I$ receives a request for maintenance with respect to expiration of Page Counter i from Status Procedure, by the signal $SchOper_i$ and forwards it to the manager. The manager confirms the scheduling with the other functions and sends a response back to the Status Procedure, using $\_ExOper_i$. It also receives feedback from Maintenance Operation that the maintenance is finished in order to reset the scheduling, again triggered by $\_OpFin_{ij}?$.

4.2 Coordination Requirements

Status Procedure adheres to several coordination rules:

1) *Maintenance operations can be performed only when Printing Process Function is in standby.* This state exclusion property requires a maintenance operation ij to be in progress, identified by $MO_{ij} = 2$, only if the printer is in standby, i.e., $CPM = 1$. Thus, we specify that the following must always hold:

$$\neg(CPM \neq 1 \wedge \bigvee_{i \in I, j \in J_i} MO_{ij} = 2) \tag{5}$$

2) *Maintenance operations can be scheduled only if soft deadline has been reached and there are no print jobs in progress, or a hard deadline is passed.* We schedule a maintenance operation ij for $j \in J_i$ using the signal $SchOper_i$ for $i \in I$. Soft and a hard deadline for Page Counter i is identified by $PC_i = 2$ and $PC_i = 3$, respectively, leading to

$$\xrightarrow{SchOper_i!?} \Rightarrow (PC_i = 2 \wedge TPM = 1) \vee PC_i = 3 \tag{6}$$

for every $i \in I$.

3) *Maintenance operations can be started only after being scheduled.* For every $j \in J_i$ and $i \in I$. Thus, we relate $OpStart_{ij}$ with the corresponding maintenance scheduler:

$$\xrightarrow{OpStart_{ij}!?} \Rightarrow MS_i = 3. \tag{7}$$

4) *The power mode of the printing process function must follow the power mode dictated by the managers, unless overridden by a pending maintenance operation.* We model this requirement separately for switching from run to standby power mode and vice versa. We can switch from run to standby if this is required by the manager, i.e., there is a new print job, and there is no need to start a maintenance operation. This is modeled as

$$\xrightarrow{Stb2Run!?} \Rightarrow TPM = 2 \wedge \bigwedge_{i \in I} MS_i \neq 3. \tag{8}$$

Contrariwise, we switch to Standby if there is no pending job or maintenance operation:

$$\xrightarrow{Run2Stb!?} \Rightarrow TPM = 1 \vee \bigvee_{i \in I} MS_i = 3. \tag{9}$$

The set of parameterized data-based coordination requirements is given by the expressions (5) – (9).

4.3 Supervisor Synthesis

For any value of the parameters for the index sets I and J_i for $i \in I$, we can instantiate a plant and synthesize a supervisor. For the sake of clarity, we illustrate the situation when there is only one maintenance procedure. We omit the unnecessary indices of the data variables. The supervisor sends the control signals upon observation of certain data assignments, which are given in the form of guards. The indices of the guards correspond to the indices of the control requirements that concern the control signal. Note that the state-exclusion requirement is treated as a global invariant, whereas no termination option of the plant is disabled. The guards have been synthesized as follows [17]:

$$g_6 \triangleq (PC = 2 \wedge TPM = 1) \vee PC = 3 \qquad g_7 \triangleq CPM = 1 \wedge MS = 3$$
$$g_8 \triangleq MS \neq 3 \wedge TPM = 2 \wedge MO \neq 2 \qquad g_9 \triangleq (MS \neq 3 \wedge TPM = 1) \vee MS = 3.$$

The supervisor has the syntax form restricted by (1) and it is given by:

$$S \triangleq \Big(g_6 :\rightarrow SchOper!.1 + g_7 :\rightarrow OpStart!.1 + g_8 :\rightarrow Stb2Run!.1 + g_9 :\rightarrow Run2Stb!.1 + 1\Big)^*.$$

To illustrate the process of supervision, we consider the event $Stb2Run$. It is not difficult to deduce, e.g., that initially the event *Stb2Run* is not enabled since then all variables are assigned the value of 1. This corresponds to the situation where there are not print jobs waiting to be executed, so there is no reason to turn the power of the printer on. Similarly, a maintenance operation can be started only if the printer is in standby mode, identified by $CPM = 1$, and the operation has been successfully scheduled, identified by $MS = 3$.

5 Concluding Remarks

We developed a process theory encompassing communicating processes with data and generic communication actions. We applied the developed theory to model supervisory control feedback loops with data observations, where we distinguish between the observation and control flow of information. We classified the processes modeling the unsupervised system and the supervisory controller to capture their specific roles. To capture the notion of controllability, which identifies the set of feasible supervisory controllers, we employed the behavioral relation partial bisimulation and we extended the notion for the new setting. We casted the process of supervisory controller synthesis in a model-based systems engineering framework, for which implementation we employ state-of-the-art tools. To illustrate our approach, we reiterated on an industrial study dealing with coordination of maintenance procedures in a printing process of a high-tech printer. We demonstrated that our approach is capable of successfully modeling the interaction in the supervisory control loop and offers a compact representation of the model of the supervisory controller.

References

[1] K. Akesson, M. Fabian, H. Flordal & R. Malik (2006): *Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems*. In: *Proceedings of WODES 2006*, IEEE, pp. 384 – 385, doi:10.1109/WODES.2006.382401.

[2] J. C. M. Baeten, T. Basten & M. A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*. *Cambridge Tracts in Theoretical Computer Science* 50, Cambridge University Press.

[3] J. C. M. Baeten, D. A. van Beek, B. Luttik, J. Markovski & J. E. Rooda (2011): *A Process-Theoretic Approach to Supervisory Control Theory*. In: *Proceedings of ACC 2011*, IEEE, pp. 4496–4501.

[4] J. C. M. Baeten, J. M. van de Mortel-Fronczak & J. E. Rooda (2011): *Integration Of Supervisory Control Synthesis In Model-Based Systems Engineering*. In: *Proceedings of ETAI/COSY 2011*, IEEE, pp. 167 – 178.

[5] J.C.M. Baeten, D.A. van Beek, A.C. van Hulst & J. Markovski (2011): *A Process Algebra for Supervisory Coordination*. In: *Proceedings of PACO 2011*, *Electronic Proceedings in Theoretical Computer Science* 60, Open Publishing Association, pp. 36–55, doi:10.4204/EPTCS.60.3.

[6] J.C.M. Baeten & J.A. Bergstra (1997): *Process algebra with propositional signals*. *Theoretical Computer Science* 177, pp. 381–405, doi:10.1016/S0304-3975(96)00253-8.

[7] J.C.M. Baeten, A.C. van Hulst, D.A. van Beek & J. Markovski (2012): *Towards a Concurrency Theory for Supervisory Control*. SE Report 2012-01, Eindhoven University of Technology. Available at `http://se.wtb.tue.nl/sereports`.

[8] C. Cassandras & S. Lafortune (2004): *Introduction to discrete event systems*. Kluwer Academic Publishers.

[9] M. Fabian & B. Lennartson (1996): *On non-deterministic supervisory control*. *Proceedings of the 35th IEEE Decision and Control* 2, pp. 2213–2218, doi:10.1109/CDC.1996.572970.

[10] R. J. van Glabbeek (2001): *The linear time–branching time spectrum I*. *Handbook of Process Algebra* , pp. 3–99.

[11] M. Heymann & F. Lin (1998): *Discrete-Event Control of Nondeterministic Systems*. *IEEE Transactions on Automatic Control* 43(1), pp. 3–17, doi:10.1109/9.654883.

[12] C. A. R. Hoare (1978): *Communicating sequential processes*. *Commununications of the ACM* 21(8), pp. 666–677, doi:10.1145/359576.359585.

[13] N.G. Leveson (1990): *The challenge of building process-control software*. *IEEE Software* 7(6), pp. 55–62, doi:10.1109/52.60589.

[14] C. Ma & W. M. Wonham (2005): *Nonblocking Supervisory Control of State Tree Structures*. *Lecture Notes in Control and Information Sciences* 317, Springer.

[15] J. Markovski, D. A. van Beek, R. J. M. Theunissen, K. G. M. Jacobs & J. E. Rooda (2010): *A State-Based Framework for Supervisory Control Synthesis and Verification*. In: *Proceedings of CDC 2010*, IEEE, pp. 3481–3486, doi:10.1109/CDC.2010.5717095.

[16] J. Markovski, K. G. M. Jacobs, D. A. van Beek, L. J. A. M. Somers & J. E. Rooda (2010): *Coordination of Resources using Generalized State-Based Requirements*. In: *Proceedings of WODES 2010*, IFAC, pp. 300–305, doi:10.3182/20100830-3-DE-4013.00048.

[17] S. Miremadi, K. Akesson & B. Lennartson (2008): *Extraction and representation of a supervisor using guards in extended finite automata*. In: *Proceedings of WODES 2008*, IEEE, pp. 193–199, doi:10.1109/WODES.2008.4605944.

[18] M. R. Mousavi, M. A. Reniers & J. F. Groote (2005): *Notions of bisimulation and congruence formats for SOS with data*. *Information and Computation* 200(1), pp. 107–147, doi:10.1016/j.ic.2005.03.002.

[19] A. Overkamp (1997): *Supervisory Control Using Failure Semantics and Partial Specifications*. *IEEE Transactions on Automatic Control* 42(4), pp. 498–510, doi:10.1109/9.566659.

[20] P. J. Ramadge & W. M. Wonham (1987): *Supervisory Control of a Class of Discrete-Event Processes*. *SIAM Journal on Control and Optimization* 25(1), pp. 206–230, doi:10.1137/0325013.

[21] J. J. M. M. Rutten (1999): *Coalgebra, concurrency, and control*. SEN Report R-9921, Center for Mathematics and Computer Science, Amsterdam, The Netherlands.

[22] R. R. H. Schiffelers, R. J. M. Theunissen, D. A. van Beek & J. E. Rooda (2009): *Model-Based Engineering of Supervisory Controllers using CIF*. *Electronic Communications of the EASST* 21, pp. 1–10.

[23] P. Tabuada & G. J. Pappas (2006): *Linear Time Logic Control of Discrete-Time Linear Systems*. *IEEE Transactions on Automatic Control* 51(12), pp. 1862 – 1877.

[24] C. Zhou, R. Kumar & S. Jiang (2006): *Control of nondeterministic discrete-event systems for bisimulation equivalence*. *IEEE Transactions on Automatic Control* 51(5), pp. 754–765, doi:10.1109/TAC.2006.875036.